0－1背包问题的DP求解

与算法导论(第三版)章节的对应关系

- Chapter 15. 动态规划
- 涉及知识点包括：
  - 01背包问题定义
  - 01背包的动态规划求解
  - 01背包的跳跃点法求解

0 – 1背包问题

0 – 1背包问题定义

- 给定 n 个物品和一背包。物品 i 的重量是 w_i，其价值为 v_i，背包的容量为 C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大？
- 0 – 1背包问题是一个特殊的**整数规划**问题。

$$\max \sum_{i=1}^{n} v_i x_i$$

$$s.t. \begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

0 – 1背包问题 (续)

- **动态规划解法**

设所给0 – 1背包问题

$$\max \sum_{k=i}^{n} v_k x_k$$

$$s.t. \begin{cases} \sum_{k=i}^{n} w_k x_k \le j \\ x_k \in \{0,1\}, i \le k \le n \end{cases}$$

的最优值为 $m(i,j)$，即 $m(i,j)$ 是背包容量为 j，可选择物品为 $i, i+1, \cdots, n$ 时0 – 1背包问题的最优值。由0 – 1背包问题的最优子结构性质，可以建立计算 $m(i,j)$ 的递归式如下:

$$m(i,j) = \begin{cases} \max\{m(i+1,j), m(i+1,j-w_i)+v_i\} & j \ge w_i \\ m(i+1,j) & 0 \le j < w_i \end{cases}$$

0 – 1背包问题 (续)

递推式的初始化：

$$m(n,j)=\begin{cases} v_n & j \geq w_n \\ 0 & 0 \leq j < w_n \end{cases}$$

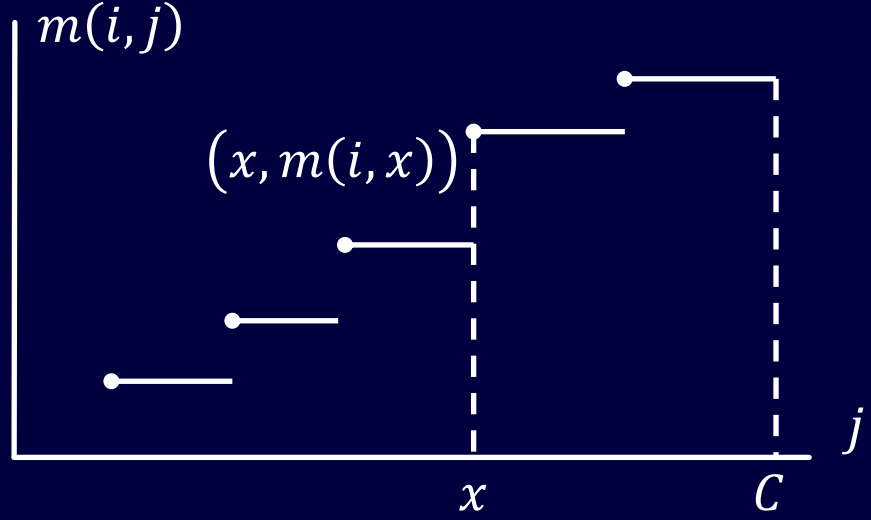
算法复杂度分析：

从 $m(i,j)$ 的递归式容易看出，算法需要 $O(nC)$ 计算时间。当背包容量 C 很大时，算法需要的计算时间较多。例如，当 $C > 2^n$ 时，算法需要 $\Omega(n2^n)$ 计算时间。

0 – 1背包问题 (续)

- **算法改进：跳跃点解法**

由 $m(i,j)$ 的递归式容易证明，在一般情况下，对每一个确定的 $i(1 \le i \le n)$，**函数 $m(i,j)$ 是关于变量j的阶梯状单调不减函数**。跳跃点是这一类函数的描述特征。在一般情况下，函数 $m(i,j)$ 由其全部跳跃点唯一确定。如图所示。

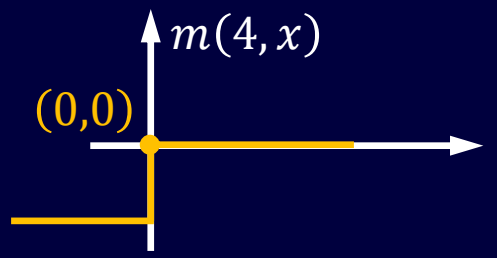


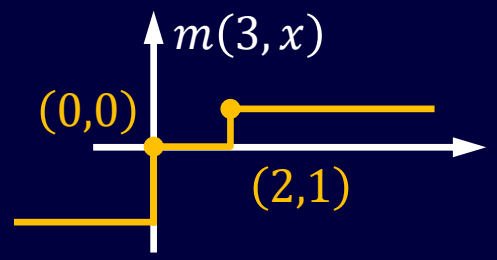
对每一个确定的 $i\ (1 \le i \le n)$，用一个表 $p[i]$ 存储函数 $m(i,j)$ 的全部跳跃点。表 $p[i]$ 可以计算 $m(i,j)$ 的递归式递归地由表 $p[i+1]$ 计算，初始时 $p[i+1] = \{(0,0)\}$

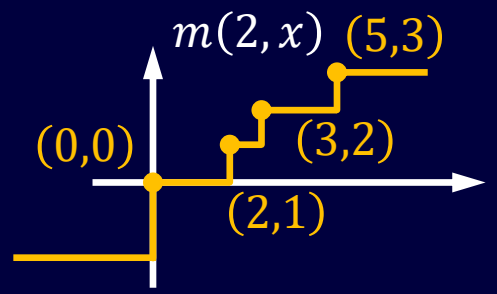
0 – 1背包问题 (续)

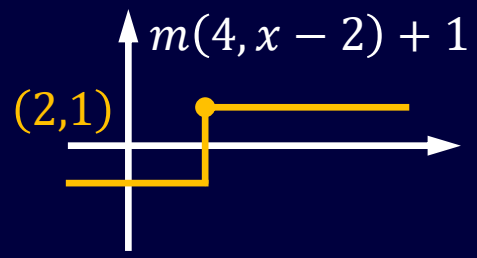
$$m(i,j)=\begin{cases}\max\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

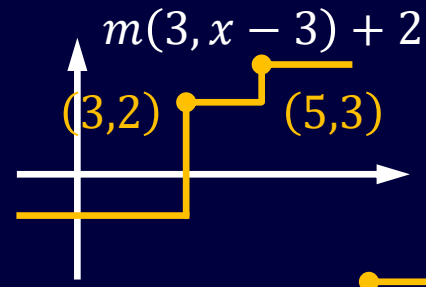
- 一个例子：$n=3$， $c=6$，$w=\{4,3,2\}$， $v=\{5,2,1\}$。

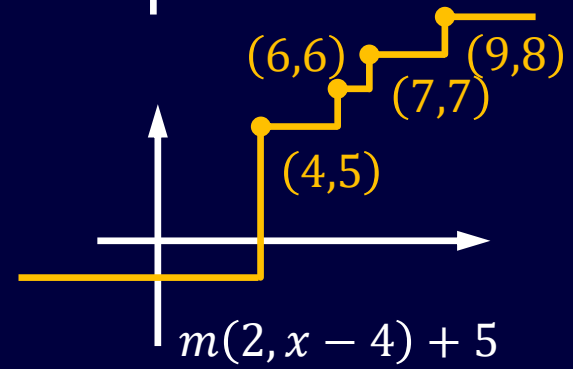


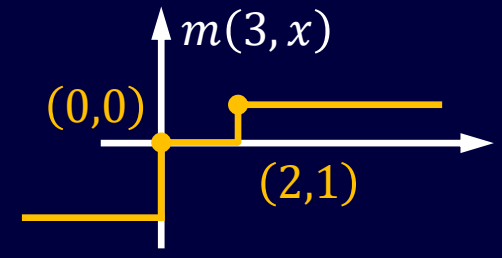


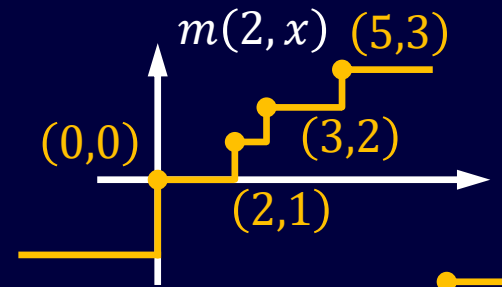


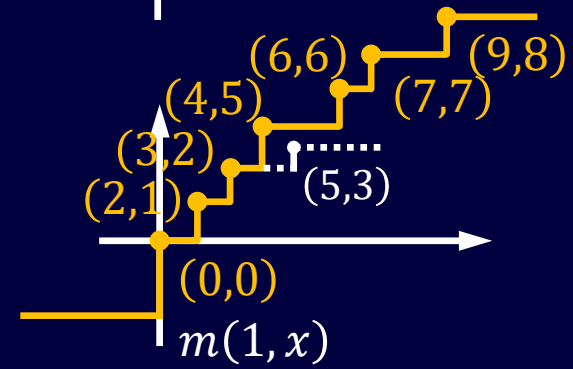












0 – 1背包问题 (续)

算法改进：跳跃点解法 (续)

- 函数 $m(i,j)$ 是由函数 $m(i+1,j)$ 与函数 $m(i+1,j-w_i)+v_i$ 作max运算得到的。因此，函数 $m(i,j)$ 的全部跳跃点包含于函数 $m(i+1,j)$ 的跳跃点集 $p[i+1]$ 与函数 $m(i+1,j-w_i)+v_i$ 的跳跃点集 $q[i+1]$ 的并集中。易知，$(s,t) \in q[i+1]$ 当且仅当 $w_i \le s \le c$ 且 $(s-w_i,t-v_i) \in p[i+1]$。因此，可以由 $p[i+1]$ 确定跳跃点集 $q[i+1]$ 如下。

$$\begin{aligned} q[i+1] &= p[i+1] \oplus (w_i, v_i) \\ &= \{(j+w_i, m(i,j)+v_i) \mid (j, m(i,j)) \in p[i+1]\} \end{aligned}$$

- 另一方面，设 (a,b) 和 (c,d) 是 $p[i+1] \cup q[i+1]$ 中的2个跳跃点，则当 $c \ge a$ 且 $d < b$ 时，(c,d) 受控于 (a,b)，从而 (c,d) 不是 $p[i]$ 中的跳跃点。除受控跳跃点外， $p[i+1] \cup q[i+1]$ 中的其它跳跃点均为 $p[i]$ 中的跳跃点。
- 由此可见，在递归地由表 $p[i+1]$ 计算表 $p[i]$ 时，可先由 $p[i+1]$ 计算出 $q[i+1]$，然后合并表 $p[i+1]$ 和表 $q[i+1]$，并清除其中的受控跳跃点得到表 $p[i]$。

0 – 1背包问题 (续)

- **一个例子：** $n=5$， $c=10$， $w=\{2,2,6,5,4\}$， $v=\{6,3,5,4,6\}$。
  - 初始时 $p[6]=\{(0,0)\}$，由 $(w_5,v_5)=(4,6)$ 有 $q[6]=p[6]\oplus(w_5,v_5)=\{(4,6)\}$。
  - 由跳跃点集 $p[6]$ 与 $q[6]$ 的并集得 $p[5]=\{(0,0),(4,6)\}$。因此，$q[5]=p[5]\oplus(w_4,v_4)=\{(5,4),(9,10)\}$。
  - 从跳跃点集 $p[5]$ 与 $q[5]$ 的并集 $p[5]\cup q[5]=\{(0,0),(4,6),(5,4),(9,10)\}$ 中可以看到跳跃点 $(5,4)$ 受控于跳跃点 $(4,6)$。将受控跳跃点 $(5,4)$ 清除后，得到 $p[4]=\{(0,0),(4,6),(9,10)\}$。同时，$q[4]=p[4]\oplus(w_3,v_3)=\{(6,5),(10,11),\sout{(15,15)}\}$。
  - $p[4]$ 与 $q[4]$ 并集且清除跳跃点后有 $p[3]=\{(0,0),(4,6),(9,10),(10,11)\}$。同时，$q[3]=p[3]\oplus(w_2,v_2)=\{(2,3),(6,9),\sout{(11,13),(12,14)}\}$。
  - $p[3]$ 与 $q[3]$ 并集且清除跳跃点后有 $p[2]=\{(0,0),(2,3),(4,6),(6,9),(9,10),(10,11)\}$。同时，$q[2]=p[2]\oplus(w_1,v_1)=\{(2,3),(4,9),(6,12),(8,15),\sout{(11,16),(12,17)}\}$。
  - $p[1]$ 的最后的那个跳跃点 $(8,15)$ 给出所求的最优值为 $m(1,c)=15$。

0 – 1背包问题 (续)

算法复杂度分析

- 上述算法的主要计算量在于计算跳跃点集 $p[i](1 \le i \le n)$。由于 $q[\mathrm{i}+1] = p[\mathrm{i}+1] \oplus (w_{\mathrm{i}}, v_{\mathrm{i}})$，故计算 $q[\mathrm{i}+1]$ 需要 $\mathrm{O}(|p[i+1]|)$ 计算时间。合并 $p[\mathrm{i}+1]$ 和 $q[\mathrm{i}+1]$ 并清除受控跳跃点也需要 $\mathrm{O}(|p[i+1]|)$ 计算时间。从跳跃点集 $p[i]$ 的定义可以看出， $p[i]$ 中的跳跃点相应于 $x_i, \cdots, x_n$ 的0/1赋值。因此，$p[i]$中跳跃点个数不超过 $2n-i+1$。由此可见，算法计算跳跃点集 $p[i]$ 所花费的计算时间为

$$O\left(\sum_{i=2}^{n} |p[i+1]|\right) = O\left(\sum_{i=2}^{n} 2^{n-i}\right) = O\left(2^n\right)。$$

从而，改进后算法的计算时间复杂性为 $\mathrm{O}(2^n)$。当所给物品的重量 $w_i(1 \le i \le n)$ 是整数时，$|p[i]| \le C+1$，$(1 \le i \le n)$。在这种情况下，改进后算法的计算时间复杂性为 $\mathrm{O}(min\{nC, 2^n\})$。